Theoretical Computer Science 97 (1992) 199–215
Elsevier

On polynomial-time Turing and many–one completeness in PSPACE*

Osamu Watanabe
Department of Computer Science, Tokyo Institute of Technology, Meguro-ku, Tokyo 152, Japan

Shouwen Tang
Beijing Computer Institute, Beijing 100044, People's Republic of China

Communicated by J. Díaz
Received May 1990
Revised June 1990

Abstract

Watanabe, O. and S. Tang, On polynomial-time Turing and many–one completeness in PSPACE, Theoretical Computer Science 97 (1992) 199–215.

Meyer and Paterson (1979) investigated $\leqslant_m^P$-reducibility of PSPACE-complete sets to sparse sets. We point out that their observation indicates the difference between the powers of $\leqslant_T^P$- and $\leqslant_m^P$-reducibilities, and establish a machinery that uses this difference for separating $\leqslant_T^P$-completeness from $\leqslant_m^P$-completeness in PSPACE. By using this machinery, we show the difference between $\leqslant_T^P$- and $\leqslant_m^P$-completeness notions in PSPACE from each of the following assumptions: (i) a randomized completeness notion differs from a deterministic one in PSPACE, and (ii) PSPACE has a "dense" set, almost every element of which is "hard" to produce by any polynomial-time computation.

1. Introduction

The notion of completeness describes the idea of being most intractable in a given complexity class. For example, a set is called NP-complete in general if every set in NP is polynomial-time deterministically reducible to the set and the set itself is in NP. Cook [7] formalized this notion considering *polynomial-time Turing reductions* ($\leqslant_T^P$-reductions), reductions via polynomial-time deterministic oracle Turing machines. He

* A part of this research was performed while the authors were visiting the Department of Mathematics, University of California, Santa Barbara, and was supported in part by the National Science Foundation under grants CCR-8611980.

showed that SAT is NP-complete by $\leqslant_T^P$-reductions. Karp [11] pointed out that the NP-completeness of SAT is provable by using much simpler reductions, i.e. *polynomial-time many-one reductions* ($\leqslant_m^P$)-reductions, reductions via polynomial-time deterministically computable functions. Since then, we have been able to prove all the NP-completeness results by using only $\leqslant_m^P$-reductions. This fact leads us to ask whether $\leqslant_T^P$-reducibility provides a stronger completeness notion than $\leqslant_m^P$-reducibility. This question is investigated in this paper.

It is known that $\leqslant_T^P$-completeness indeed differs from $\leqslant_m^P$-completeness in higher complexity classes such as E [13, 22] and NE [21] [where $E \stackrel{\text{def}}{=} \bigcup_{c>0} \text{DTIME}(2^{cn})$ and $NE \stackrel{\text{def}}{=} \bigcup_{c>0} \text{NTIME}(2^{cn})$]. It is shown [14] that Turing reductions are *polynomially faster* than many-one reductions in the class of $\leqslant_m^P$-complete sets in NP. Nevertheless, nothing, except trivial observations, has been reported concerning the difference between polynomial-time Turing and many-one completeness in polynomial complexity classes. For example, we know *by definition* that if $NP \neq \Delta_2^P$, then SAT is $\leqslant_T^P$-complete but not $\leqslant_m^P$-complete in Δ_2^P. On the other hand, we know no such results for NP, Σ_k^P and PSPACE. Here we investigate the difference in PSPACE. We consider two assumptions of different sorts, from each of which we prove the difference between $\leqslant_T^P$- and $\leqslant_m^P$-completeness in PSPACE. In this paper, *the separation* will mean the separation of $\leqslant_T^P$-completeness from $\leqslant_m^P$-completeness in PSPACE. Although we often omit "polynomial-time", all the reductions, reducibilities and completeness notions discussed in this paper should be regarded as polynomial-time.

The first separation result is proved from an assumption concerning the difference between randomized and deterministic completeness notions.

Adleman and Manders [1] introduced one notion of "polynomial-time randomized reducibility", and showed that a problem not known to be NP-complete under deterministic reducibility is complete under this reducibility. (Some other examples have been reported in [2, 20, 19].) Here we ask whether the difference between randomized and deterministic completeness implies the difference between $\leqslant_T^P$- and $\leqslant_m^P$-completeness, and present an affirmative answer in PSPACE. That is, we show that the difference between $\leqslant_T^{BPP}$- and $\leqslant_m^P$-completeness in PSPACE implies the separation. Note that $\leqslant_T^{BPP}$-reducibility is the most general randomized reducibility and that $\leqslant_m^P$-reducibility is the most restrictive deterministic reducibility. Hence, our result shows that we have the separation from any other type of difference between polynomial-time randomized and deterministic completeness in PSPACE.

The second separation result is shown from an assumption concerning the existence of a dense set in PSPACE whose elements are hard to produce.

In order to measure density of a given set, we introduce the concept of length-density. A set is called *length-dense* if for some polynomial p and all $n>0$, it has at least one element x such that $n \leqslant |x| \leqslant p(n)$. We use the *generalized Kolmogorov complexity* [10] for measuring hardness of producing a given string. We consider the following condition (K) (see Section 4 for the precise statement): there exists a length-dense set in

PSPACE almost every element of which has "high" generalized Kolmogorov complexity. We show that (K) implies the separation.

Although condition (K) is rather technical, we will see that it has an interesting interpretation, which yields its sufficient conditions. We show that condition (K) is closely related to notions concerning "almost everywhere complexity" for "problems of computing function values". For "decision problems", the concept of *polynomial complexity core* has been used to study almost everywhere complexity (see e.g. [6]). We generalize this concept for "problems of computing function values"; that is, we introduce the *polynomially hard domain*, which corresponds to the polynomial complexity core, in order to discuss almost everywhere complexity of computing functions. It is shown that (K) is equivalent to the existence of a polynomial space computable function whose output length is polynomially bounded and that has a length-dense polynomially hard domain in PSPACE. From this interpretation, we can prove that each of the following two conditions implies (K): (i) $\mathrm{DTIME}(t) \subseteq \mathrm{PSPACE}$ for some super-polynomial running time t, and (ii) there exists a one-way function that satisfies "desired" requirements for cryptosystems (see Section 4.3 for the precise statement). Thus, each of these two conditions also implies the separation.

Sparse sets have received considerable attention in structural complexity theory; in particular, several observations have been reported concerning polynomial-time reducibility to sparse sets [16, 17, 23]. We use one such observation for proving our separation results. Let U be any $\leqslant^{\mathrm{P}}_{\mathrm{m}}$-complete set in PSPACE. Meyer and Paterson [17] proved that for every sparse set S, $U \leqslant^{\mathrm{P}}_{\mathrm{m}} S$ implies $U \in P$. A relativized version of their argument proves that for every set C and sparse set S, $U \leqslant^{\mathrm{P}}_{\mathrm{m}} C \oplus S$ implies $U \leqslant^{\mathrm{P}}_{\mathrm{T}} C$. This indicates the difference between the power of $\leqslant^{\mathrm{P}}_{\mathrm{T}}$- and $\leqslant^{\mathrm{P}}_{\mathrm{m}}$-reducibilities; that is, an easily separable sparse part (i.e. S) of a $\leqslant^{\mathrm{P}}_{\mathrm{m}}$-complete set (i.e. $C \oplus S$) can be deleted when $\leqslant^{\mathrm{P}}_{\mathrm{T}}$-reductions are used in place of $\leqslant^{\mathrm{P}}_{\mathrm{m}}$-reductions. Indeed, from this observation, we have the following key lemma:

Key Lemma. *If there exists a set C in* PSPACE *and a sparse set S in* PSPACE *such that $U \not\leqslant^{\mathrm{P}}_{\mathrm{m}} C$ and $U \leqslant^{\mathrm{P}}_{\mathrm{T}} C \oplus S$, then $\leqslant^{\mathrm{P}}_{\mathrm{T}}$-completeness differs from $\leqslant^{\mathrm{P}}_{\mathrm{m}}$-completeness in* PSPACE.

We show that each assumption stated above yields C and S satisfying this lemma.

2. Preliminaries

In this paper we use the alphabet $\Sigma = \{0, 1\}$. By a *language* we mean a subset of Σ^*. Nonnegative integers are embedded into Σ^* by a binary encoding. We denote by $|x|$ the length of a string x, and by $\| X \|$ the cardinality of X. Let $A^{\leqslant n}$, $A^{=n}$ and $A^{n \leqslant \cdot \leqslant m}$ denote $\{x \in A : |x| \leqslant n\}$, $\{x \in A : |x| = n\}$ and $\{x \in A : n \leqslant |x| \leqslant m\}$, respectively. For any sets A and B, $A \oplus B$ is defined as $\{0x, 1y : x \in A, y \in B\}$. We use A^{c} to denote the complement

of A, i.e. $\Sigma^* - A$. We consider a standard one-to-one pairing function from $\Sigma^* \times \Sigma^*$ to Σ^*; for inputs x and y, we denote the output of the pairing function by $\langle x, y \rangle$. Without loss of generality, we can assume that our pairing function is polynomial-time invertible and that for some constant c_0, $\forall x, y[|\langle x, y \rangle| \leqslant c_0(|x|+|y|)]$. A *measure function* is a total function from nonnegative integers to nonnegative integers. In what follows, by a *function* we mean a (not necessarily total) function from Σ^* to Σ^* unless otherwise indicated. We use $Dom(f)$ to denote the domain of a function f.

We follow standard definitions and notations of computational complexity theory (see e.g. [5]). Out computational model is the standard multitape Turing machine acceptor and transducer. An acceptor is deterministic, nondeterministic, or randomized (i.e. probabilistic with bounded error probability); on the other hand, a transducer is always deterministic. An acceptor is denoted by M, and a transducer is denoted by N. Let $L(M)$ denote the set of strings accepted by M. For any string x, we say that N *on* x *yields an output* y if N on input x enters an accepting state with y on its output tape. We use N to denote the function computed by N; that is, for every x, $N(x)$ is the output that N yields on input x. Note that $N(x)$ is undefined if N on x never enters an accepting state. A function f is called *polynomial-time computable* (*polynomial-space computable*) if f is computed by some polynomial-time bounded (polynomial-space bounded) transducer N. The above definitions are relativized in a usual way. Let $L(M, A)$ denote the set of strings accepted by an oracle machine M relative to A. For any set A, let $\mathrm{P}(A)$ denote the class of languages accepted by polynomial-time bounded deterministic oracle machines relative to A; $\mathrm{NP}(A)$ and $\mathrm{BPP}(A)$ are defined similarly.

Definition 2.1. Let A and B be any sets of strings.

(1) A is $\leqslant_{\mathrm{m}}^{\mathrm{P}}$-*reducible* to B ($A \leqslant_{\mathrm{m}}^{\mathrm{P}} B$) if there exists a polynomial-time computable total function f such that $x \in A$ iff $f(x) \in B$.

(2) A is $\leqslant_{\mathrm{T}}^{\mathrm{P}}$-*reducible* to B ($A \leqslant_{\mathrm{T}}^{\mathrm{P}} B$) if $A \in \mathrm{P}(B)$.

(3) A is $\leqslant_{\mathrm{T}}^{\mathrm{BPP}}$-*reducible* to B ($A \leqslant_{\mathrm{T}}^{\mathrm{BPP}} B$) if $A \in \mathrm{BPP}(B)$.

Remark. We also use a relativized version of (1). For any sets A, B, and C, A is $\leqslant_{\mathrm{m}}^{\mathrm{P}(C)}$-*reducible* to B ($A \leqslant_{\mathrm{m}}^{\mathrm{P}(C)} B$) if there exists a total function f, which is polynomial-time computable relative to C, such that $x \in A$ iff $f(x) \in B$.

In order to simplify statements, we will abuse the notation of reducibilities in the following way: for any class of languages $\mathscr{C}$ and any reduction type $\leqslant_r^{\mathrm{P}}$, we write $\mathscr{C} \leqslant_r^{\mathrm{P}} A$ if every set in $\mathscr{C}$ is $\leqslant_r^{\mathrm{P}}$-reducible to A. A set A is called $\leqslant_r^{\mathrm{P}}$-*hard for* $\mathscr{C}$ if $\mathscr{C} \leqslant_r^{\mathrm{P}} A$, and B is called $\leqslant_r^{\mathrm{P}}$-*complete in* $\mathscr{C}$ if B is $\leqslant_r^{\mathrm{P}}$-hard for $\mathscr{C}$ and $B \in \mathscr{C}$.

For any $n \geqslant 0$, let *first*$(A; n)$ denote the smallest element of A of length n by the lexicographic ordering. Note that *first*$(A; n)$ is undefined if A has no element of length

n. We define a *prefix set* of A, $Pref(A)$, and a *first set* of A, $First(A)$, as follows:

$$Pref(A)=\{\langle 0^n, w\rangle : w \text{ is a prefix of some } x\in A^{=n}\}; \text{ and}$$

$$First(A)=\{first(A;n): n\geqslant 0\}.$$

We often consider a set $Pref(First(A))$ because, for each n, one can easily produce one element of $A^{=n}$ (if it exists) relative to $Pref(First(A))$. That is, there is an oracle transducer that produces, for a given 0^n, the first element of $A^{=n}$ within polynomial time relative to $Pref(First(A))$.

In order to simplify statements, we use the following abbreviations: for any set A and any predicate R,

$$\forall^{\infty} x\in A[R(x)] \Leftrightarrow [R(x) \text{ for all but finitely many elements in } A]; \text{ and}$$

$$\exists^{\infty} x\in A[R(x)] \Leftrightarrow [R(x) \text{ for infinitely many elements in } A].$$

3. Key lemma and the first separation result

In this section we establish a machinery for proving the separation, i.e. the difference between $\leqslant_T^P$- and $\leqslant_m^P$-completeness in PSPACE. Then we prove the first main result: we show that the difference between $\leqslant_T^{BPP}$- and $\leqslant_m^P$-completeness implies the separation.

We first recall the following well-known properties of PSPACE.

Proposition 3.1.

(1) PSPACE *is closed under complementation. That is, for any set* $A\in$PSPACE, $A^c\in$PSPACE.

(2) PSPACE *is closed under* NP( )*-operation. That is, for any set* $A\in$PSPACE, $NP(A)\subseteq$PSPACE.

(3) *There is a self-reducible* $\leqslant_m^P$*-complete set in* PSPACE (see e.g. [12, 17]).

We will see that the following theorem provides a useful tool – Lemma 3.4 – for distinguishing $\leqslant_T^P$-completeness from $\leqslant_m^P$-completeness.

Theorem 3.2 (A relativized version of the theorem by Meyer and Paterson [17]). *Let* A *be any self-reducible set. For any sparse set* S *and any set* B, *if* $A\leqslant_m^{P(B)} S$ *and* $A^c\leqslant_m^{P(B)} S$, *then* $A\in P(B)$.

Proof. Meyer and Paterson [17, Theorem 3] proved that for every self-reducible set L, if both L and L^c are $\leqslant_m^P$-reducible to some sparse set, then L is in P. It is easy to see

that their proof can be relativized as stated in the theorem. The detail is left to the interested reader. □

Corollary 3.3. *For any sparse set S and any set C, if* $\mathrm{PSPACE} \leqslant_m^P C \oplus S$, *then* $\mathrm{PSPACE} \leqslant_T^P C$.

Proof. We use the properties stated in Proposition 3.1. Let A be a self-reducible $\leqslant_m^P$-complete set in PSPACE. Since both A and A^c are in PSPACE, they are $\leqslant_m^P$-reducible to $C \oplus S$; thus, A, $A^c \leqslant_m^{P(C)} S$. (Here we are assuming without loss of generality that S is nonempty.) Then Theorem 3.2 shows that $A \in \mathrm{P}(C)$, i.e. $A \leqslant_T^P C$. □

This corollary suggests the difference between the power of $\leqslant_T^P$- and $\leqslant_m^P$-reducibilities; that is, an easily separable sparse part (i.e. S) of a $\leqslant_m^P$-complete set (i.e. $C \oplus S$) can be deleted while a $\leqslant_m^P$-reduction is replaced with a $\leqslant_T^P$-reduction. Our key lemma, stated below, makes use of this difference.

Lemma 3.4. *Suppose that there exists a set C and a sparse set S such that*

(i) *both C and S are in* PSPACE;

(ii) *C is not $\leqslant_m^P$-hard for* PSPACE; *and*

(iii) *$C \oplus S$ is $\leqslant_T^P$-hard for* PSPACE.

Then there exists a $\leqslant_T^P$-complete set in PSPACE *that is not $\leqslant_m^P$-complete in* PSPACE.

Proof. Let C and S be sets satisfying (i)–(iii). Suppose that C is $\leqslant_T^P$-hard for PSPACE; then we are done since it is clear that C satisfies our purpose. Now suppose that C is not $\leqslant_T^P$-hard for PSPACE. We show that $C \oplus S$ is the desired set. Note that $C \oplus S$ is $\leqslant_T^P$-complete in PSPACE [from (i) and (iii)]. On the other hand, if $C \oplus S$ is $\leqslant_m^P$-hard for PSPACE, then it follows from Corollary 3.3 that C is $\leqslant_T^P$-hard for PSPACE; a contradiction. Hence, $C \oplus S$ is not $\leqslant_m^P$-hard (so, not $\leqslant_m^P$-complete) for PSPACE. □

Now we apply this lemma to obtain the first separation result.

As mentioned in Section 1, we have some candidates that may witness the difference between randomized and deterministic completeness notions. Thus, the question of interest is whether such difference between randomized and deterministic completeness implies the difference between $\leqslant_T^P$- and $\leqslant_m^P$-completeness. The following theorem provides an affirmative answer to this question in PSPACE.

Theorem 3.5. *Consider* PSPACE. *Assume that there exists a $\leqslant_T^{BPP}$-complete set that is not $\leqslant_m^P$-complete. Then there exists a $\leqslant_T^P$-complete set that is not $\leqslant_m^P$-complete.*

Remark. Note that $\leqslant_m^P$-completeness is the strongest deterministic completeness and that $\leqslant_T^{BPP}$-completeness is the weakest randomized completeness. Thus, for example,

every set that is $\leqslant_T^R$-complete but not $\leqslant_{tt}^P$-complete in PSPACE also witnesses the difference between $\leqslant_T^{BPP}$- and $\leqslant_m^P$-completeness notions in PSPACE. [Here the $\leqslant_T^R$-reducibility is defined as follows: $A \leqslant_T^R B$ if $A \in R(B)$.] Therefore, the above theorem gives an affirmative answer to our question considering the most general case.

Proof. Let C be a set satisfying the assumption; namely, $C \in$PSPACE, PSPACE $\leqslant_T^{BPP} C$ and PSPACE $\not\leqslant_m^P C$. In order to use Lemma 3.4, it suffices to show a sparse set S in PSPACE such that PSPACE $\leqslant_T^P C \oplus S$.

Let U be any $\leqslant_m^P$-complete set in PSPACE. Clearly, $U \in \mathrm{BPP}(C)$ since $U \in$PSPACE and PSPACE $\leqslant_T^{BPP} C$. Hence, it follows from Lemma 3.6 that there exists a sparse set S in PSPACE such that $U \in \mathrm{P}(C \oplus S)$. Therefore, PSPACE $\leqslant_T^P C \oplus S$. □

Lemma 3.6. *Let A and B be any sets in* PSPACE *such that $A \in \mathrm{BPP}(B)$. Then there exists a sparse set S in* PSPACE *such that $A \in \mathrm{P}(B \oplus S)$.*

Proof. It is well known (see e.g. [5]) that for every set L in BPP, there exists a tally set T such that $L \in \mathrm{P}(T)$. Considering its relativization, we obtain that for every set L in $\mathrm{BPP}(B)$, there exists a tally set T such that $L \in \mathrm{P}(B \oplus T)$. Thus, there exists a tally set T such that $A \in \mathrm{P}(B \oplus T)$. Let M be a polynomial-time deterministic oracle machine such that $A = L(M, B \oplus T)$. Let p_M be a polynomial time bound of M, where we can assume that p_M is an increasing function. Note that M on input x does not query a string of length $> p_M(|x|)$.

We consider the following encoding of tally sets over $\{0, 1\}^*$. For any $\tau \in \{0, 1\}^*$, let T_τ denote the following tally set: $T_\tau = \{0^i$: the ith bit of τ is $1\}$. Thus, for every tally set T and $m \geqslant 0$, there exists a string $\tau \in \{0, 1\}^*$ that encodes $T^{\leqslant m}$, i.e. $T_\tau = T^{\leqslant m}$.

Since $A = L(M, B \oplus T)$ for some tally set T, we have

$$(*1) \qquad \forall n \geqslant 0,\ \exists \tau \in \{0, 1\}^{p_M(n)},\ \forall x \in \Sigma^n [x \in A \Leftrightarrow x \in L(M, B \oplus T_\tau)].$$

Define $W = \bigcup_{n \geqslant 0} \{\tau \in \{0, 1\}^{p_M(n)}$: τ satisfies $(*1)$ w.r.t. $n\}$. Note that both A and B are in PSPACE; hence, it follows from Proposition 3.1 (1) and (2) that a polynomial space bounded machine can check $(*1)$ for every τ. Thus, W is in PSPACE.

Now define $S = \mathit{Pref}(\mathit{First}(W))$. Clearly, S is sparse, and it follows from Proposition 3.1 (1) and (2) that S is in PSPACE. By a binary search relative to S, one can easily produce a string τ of length $p_M(n)+1$ satisfying $(*1)$. This proves that $A \in P(B \oplus S)$. □

4. The second separation result

In this section we show the separation, i.e. the difference between $\leqslant_T^P$- and $\leqslant_m^P$-completeness in PSPACE, from an assumption concerning the existence of a "dense" set in PSPACE whose elements are "hard" to produce.

In order to measure the difficulty of producing a string, we use the concept of generalized Kolmogorov complexity [10]. Let N_u denote some standard universal Turing transducer that can simulate any universal Turing machine efficiently; more precisely, N_u is a machine that satisfies Fact 1 in [10]. For any measure functions δ and t, a *generalized Kolmogorov complexity class* $\mathrm{K}[\delta, t]$ is defined as follows:

$$\mathrm{K}[\delta, t] = \{y: \exists x: |x| \leqslant \delta(|y|)\ [N_u \text{ on input } x \text{ yields } y \text{ within } t(|y|) \text{ steps}]\}.$$

That is, $\mathrm{K}[\delta, t]$ is the set of strings y generated by N_u from a string of length $\leqslant \delta(|y|)$ within $t(|y|)$ steps.

Our assumption is stated precisely as the following condition (K):

(K) There exists a set X in PSPACE such that
(i) $\exists d > 0, \forall p$: polynomial $\forall^{\infty} x \in X\,[x \notin \mathrm{K}[n^d, p(n)]]$, and
(ii) $\exists q$: polynomial, $\forall n \geqslant 0, \exists x\,[n \leqslant |x| \leqslant q(n) \wedge x \in X]$.

In the next section we prove the separation from this condition. The above two properties (i) and (ii) will be referred as (K-i) and (K-ii) in the later discussions.

Although condition (K) is stated in a rather technical way, we can explain its meaning and its possibility in a more intuitive way. In Section 4.2, we introduce the notion of "polynomially hard domain", which is used to express the "almost everywhere hard part" for computing a given function. Then we show a close connection between (K) and the existence of a function with a polynomially hard domain. From this observation, condition (K) is restated as follows: there exists a polynomial-time–space computable honest function whose domain has a length-dense "hard part" (i.e. polynomially hard domain) that is PSPACE recognizable. In Section 4.3 we study the possibility that (K) holds, and show the following two sufficient conditions for (K): (i) $DTIME(t) \subseteq$ PSPACE for some super-polynomial running time t, and (ii) there exists a one-way function that satisfies "desired" requirements for cryptosystems.

4.1. Proof of the second separation result

We prove that condition (K) implies the separation. In the proof, we make use of the following two lemmas.

Lemma 4.1. *Suppose that condition* (K) *holds. Then there exists a set* Y *such that*
(i) Y *is a sparse set in* PSPACE;
(ii) Y *satisfies* (K); *and*
(iii) $Pref(Y)$ *is in* PSPACE $-$ P.

Proof. Let X be a set witnessing condition (K). Define $Y = First(X)$; we show that Y is the desired set. It is clear that $Y\,(= First(X))$ inherits the property (K-i) and (K-ii) of X; thus, Y satisfies (ii). From Proposition 3.1, we have that both Y and $Pref(Y)$ are in PSPACE. Now it suffices to prove that $Pref(Y) \notin$ P. Suppose otherwise; then it is easy

to see that $Y \subseteq \mathrm{K}[d \log n + d, n^c + c]$ for some $d > 0$ and $c > 0$. (See also e.g. [4].) This contradicts the property (K-i) of Y. □

Lemma 4.2. *Condition* (K) *implies that no* $\leqslant^{\mathrm{P}}_{\mathrm{m}}$*-complete set in* PSPACE *is* $\leqslant^{\mathrm{P}}_{\mathrm{ctt}}$*-reducible to a sparse set.*

Proof. By Lemma 4.1(iii), (K) implies $\mathrm{P} \neq \mathrm{PSPACE}$. On the other hand, Ukkonen [18] proved that $\mathrm{P} \neq \mathrm{PSPACE}$ implies that no $\leqslant^{\mathrm{P}}_{\mathrm{m}}$-complete set in PSPACE is $\leqslant^{\mathrm{P}}_{\mathrm{ctt}}$-reducible to a sparse set. Thus, we have the lemma. □

Theorem 4.3. *Condition* (K) *implies that* $\leqslant^{\mathrm{P}}_{\mathrm{T}}$*-completeness differs from* $\leqslant^{\mathrm{P}}_{\mathrm{m}}$*-completeness in* PSPACE.

Proof. Suppose that condition (K) holds. Then from Lemma 4.1, there is a sparse set Y in PSPACE having property (K-i) and (K-ii); let d and q, respectively, be a constant and a polynomial witnessing that Y satisfies (K-i) and (K-ii).

Let U denote a standard $\leqslant^{\mathrm{P}}_{\mathrm{m}}$-complete set in PSPACE. Define C by

$$C = \{\langle x, y\rangle : (\mathrm{i})\, m \leqslant |y| \leqslant q(m), \text{ where } m = 2|x|^{1/d}, \text{ and } (\mathrm{ii})\, x \in U \wedge y \in Y\}.$$

(*Note*: The coefficient "2" used here is not essential. It can be any real number $r > 1$.) Then it is easy to show that C is in PSPACE. Note that for every x, there is some y_x in Y such that $m \leqslant |y_x| \leqslant q(m)$, where $m = 2|x|^{1/d}$; thus, $x \in U \Leftrightarrow \langle x, y_x\rangle \in C$. That is, if for each x, one can find such y_x in polynomial time, then a standard complete set U is reducible to C, which is possible if $\mathit{Pref}(Y)$ is provided. Hence, $C \oplus \mathit{Pref}(Y)$ is $\leqslant^{\mathrm{P}}_{\mathrm{T}}$-hard (so $\leqslant^{\mathrm{P}}_{\mathrm{T}}$-complete) for PSPACE.

Note that $\mathit{Pref}(Y)$ is sparse since Y is sparse. Therefore, the theorem follows from the claim below and Lemma 3.4. □

Claim. *The set* C *is not* $\leqslant^{\mathrm{P}}_{\mathrm{m}}$*-hard for* PSPACE.

Proof. Assume to the contrary that C is $\leqslant^{\mathrm{P}}_{\mathrm{m}}$-hard for PSPACE; then U is $\leqslant^{\mathrm{P}}_{\mathrm{m}}$-reducible to C. We will show that then U is $\leqslant^{\mathrm{P}}_{\mathrm{ctt}}$-reducible to the sparse set Y, contradicting Lemma 4.2.

Let f be a $\leqslant^{\mathrm{P}}_{\mathrm{m}}$-reduction from U to C. Without loss of generality, we can assume that for every $z \in \Sigma^*$, $f(z)$ is of the form $\langle x, y\rangle$ for some x and y. Let f_2 be a function that maps every z to the second component of $f(z)$.

For any $z \in \Sigma^*$, let x_z and y_z be strings such that $f(z) = \langle x_z, y_z\rangle$. We first show that $\forall^{\infty} z \in U\, [|x_z| < |z|]$. Suppose otherwise; i.e. $\exists^{\infty} z \in U\, [|z| \leqslant |x_z|]$. Note that $z \in U$ implies $\langle x_z, y_z\rangle \in C$ and that $\langle x_z, y_z\rangle \in C$ implies $2|x_z|^{1/d} \leqslant |y_z| \wedge y_z \in Y$. Thus, we have

$$\exists^{\infty} z \in U\, [(2|z|)^{1/d} \leqslant (2|x_z|)^{1/d} \leqslant |y_z| \wedge y_z (= f_2(z)) \in Y].$$

Hence,

$$\exists^{\infty} y \in Y\ \exists z\ [|z| \leqslant \tfrac{1}{2}|y|^d \wedge y = f_2(z)].$$

Here note that f_2 is polynomial-time computable; thus, it follows from the above that some polynomial p exists such that $\exists^{\infty} y \in Y\ [y \in \mathrm{K}[n^d, p(n)]]$. This contradicts our assumption that the constant d witnesses the property (K-i) of Y. Therefore, there exists some $c_0 > 0$ such that

(*2) $\quad \forall z \in U\,[|z| \geqslant c_0 \Rightarrow |x_z| < |z|]$.

Now consider any string x_0, and let $n = |x_0|$. Define sequences $x_1, x_2, \ldots, x_{n-1}$ and $y_1, y_2, \ldots, y_{n-1}$ by $\langle x_{i+1}, y_{i+1} \rangle = f(x_i)$. (Note that these sequences are polynomial-time computable from x_0.) Then for any k, $1 \leqslant k < n$, we have the following relation:

$$\begin{aligned}
x_0 \in U &\Leftrightarrow f(x_0) = \langle x_1, y_1 \rangle \in C \\
&\Leftrightarrow R_1(x_1, y_1) \wedge y_1 \in Y \wedge x_1 \in U \\
&\Leftrightarrow R_1(x_1, y_1) \wedge y_1 \in Y \wedge f(x_1) = \langle x_2, y_2 \rangle \in C \\
&\Leftrightarrow R_1(x_1, y_1) \wedge y_1 \in Y \\
&\quad \wedge R_1(x_2, y_2) \wedge y_2 \in Y \wedge x_2 \in U \\
&\Leftrightarrow \quad \vdots \\
&\Leftrightarrow R_1(x_1, y_1) \wedge y_1 \in Y \\
&\quad \wedge R_1(x_2, y_2) \wedge y_2 \in Y \\
&\quad \vdots \\
&\quad \wedge R_1(x_k, y_k) \wedge y_k \in Y \wedge x_k \in U
\end{aligned}$$

(where by $R_1(x, y)$ we mean that x and y satisfy (i) in the definition of C). Thus, for every k, $1 \leqslant k < n$, we have

$$(*3) \quad x_0 \in U \Leftrightarrow \bigwedge_{1 \leqslant i \leqslant k} R_1(x_i, y_i) \wedge \bigwedge_{1 \leqslant i \leqslant k} y_i \in Y \wedge x_k \in U.$$

Assume that x_0 is in U. Then from (*3) every x_i is in U; hence, from (*2) we have $|x_0| > |x_1| > |x_2| \cdots$ until $|x_{k_0}| < c_0$ for some k_0. Here note that $k_0 < n$. Thus,

(*4) $\quad x_0 \in U \Rightarrow \exists k_0,\ 1 \leqslant k_0 < n\ [|x_{k_0}| < c_0]$.

Now from (*3) and (*4), we can conclude that $x_0 \in U$ if and only if (a) $\wedge$ (b) $\wedge$ (c) $\wedge$ (d) for the following (a), (b), (c) and (b):

(a) $\exists k_0,\ 1 \leqslant k_0 < n\ [|x_{k_0}| < c_0]$, (b) $\bigwedge_{1 \leqslant i \leqslant k_0} R_1(x_i, y_i)$,

(c) $x_{k_0} \in U$, (d) $\bigwedge_{1 \leqslant i \leqslant k_0} y_i \in Y$.

Note that (a) and (b) are polynomial-time decidable, (c) is polynomial-time decidable if $|x_{k_0}|<c_0$ and (d) is decidable by asking conjunctive queries to Y. Therefore, U is $\leqslant^{\mathrm{P}}_{\mathrm{ctt}}$ reducible to Y. A contradiction. □

4.2. Characterization of (K)

One may notice a similarity between property (K-i) and "immunity" notions, notions that have been used to express the idea of "almost everywhere hardness" (see e.g. [6]). However, ordinary immunity notions are considered for "decision problems", whereas we will discuss "problems of computing values of a function". Thus, we first introduce a concept that is appropriate for our purpose. It should be mentioned here that our approach is an alternative to the one in [9] where almost everywhere complexity of functions is also discussed.

Lynch [15] introduced the concept of polynomial complexity core, which closely relates immunity notions [6]. For any set A, an infinite set C is called a *polynomial complexity core* of A if it satisfies the following:

$$\forall p\text{: polynomial, } \forall M\text{: deterministic machine that accepts } A,$$
$$\forall^{\infty} x\in C\,[M\text{'s running time on } x \text{ exceeds } p(|x|) \text{ steps}].$$

Here we define a similar concept for "problems of computing functions".

For any measure function δ, we say that N *δ-computes* f if $\forall x\in\Sigma^*$, $\exists z$: $|z|\leqslant\delta(|x|)$ $[N(\langle x, z\rangle)=f(x)]$. A function f is called *δ-computable* if f is δ-computed by some N. Note that f is 0-computable if and only if it is partial recursive.

Definition 4.4. For any function f and any measure function δ, an infinite set $D\subseteq Dom(f)$ is a *(δ, poly)-hard domain* of f if it satisfies the following:

$$\forall p\text{: polynomial, } \forall N\text{: transducer that } \delta\text{-computes } f,$$
$$\forall^{\infty} x\in D,\ \forall z\text{: } |z|\leqslant\delta(|x|)\ \wedge\ N(\langle x, z\rangle)=f(x)$$
$$[N\text{'s running time on } \langle x, z\rangle \text{ exceeds } p(|\langle x, z\rangle|) \text{ steps}].$$

Intuitively, an infinite set $D\subseteq Dom(f)$ is a (δ, *poly*)-hard domain of f if for any polynomial p, every transducer needs more than $p(|\langle x, z\rangle|)$ steps to compute $f(x)$ for all but finitely many x in D even if it is given $\delta(|x|)$-bits of additional information. By comparing the above definitions, one can easily see the similarity of "polynomial complexity core" and "(δ, *poly*)-hard domain". Indeed, the latter notion is a generalization of the former one in the following sense.

For any set A, let c_A denote the characteristic function of A.

Proposition 4.5. *For every recursive set A, an infinite set C is a polynomial complexity core of A if and only if it is a* (0, *poly*)*-hard domain of c_A.*

(The proof is immediate from the definition; thus, it is omitted.)

For any function f, a set D is called a *polynomially hard domain* of f if it is a $(n^d, poly)$-hard domain of f for some $d>0$.

Now we state condition (K) in terms of the difficulty of functions. A function f is called *polynomially honest* (or, *honest* in short) if there exists a polynomial p such that $\forall x\in Dom(f)\ [|f(x)|\leqslant p(|x|) \wedge |x|\leqslant p(|f(x)|)]$. Intuitively, an honest function increases or decreases length by no more than a polynomial. If f is not honest, then either f or f^{-1} is not polynomial-time computable, because no polynomial time bound is enough even to print out an output. In order to avoid such trivial cases, we will consider only honest functions.

The following relation is immediate from the definition.

Proposition 4.6. *Let f be a recursive and honest function from 0^* to Σ^*. Then for every infinite set $D\subseteq Dom(f)$, the following two statements are equivalent:*

(i) *D is a polynomially hard domain of f; and*

(ii) *$f(D)$ satisfies* (K-i).

Proof. (i)→(ii): Let d_1 and d_f be constants such that D is a $(n^{d_1}, poly)$-hard domain of f, and $\forall x\in Dom(f)\ [|f(x)|\leqslant|x|^{d_f}]$. By way of contradiction, assume that $f(D)$ does not satisfy (K-i). Then there exists p_1 such that $\exists^\infty y\in f(D)\ [y\in \mathrm{K}[n^{d_1/2d_f}, p_1(n)]]$; in other words, $\exists^\infty y\in f(D)$, $\exists z{:}|z|\leqslant|y|^{d_1/2d_f}$ [N_u on z yields y within $p_1(|y|)$ steps]. Since f is recursive, there exists a transducer N_f that computes f.

Now define N_1 as follows

begin
  input($\langle x, z\rangle$);
  if $|x|\leqslant c_1$ **then** obtain $f(x)$ from a finite table and output it;
  if z is an empty string **then** {
    simulate N_f on x;
    output $N_f(x)$ (if defined)}
  else {
    simulate N_u on z', where $z=az'$ for some $a\in\Sigma$;
    output $N_u(z')$ (if defined)}
end,

where c_1 is a constant such that $\forall x\in Dom(f)\ [|x|\geqslant c_1 \Rightarrow |f(x)|^{d_1/2d_f}+1\leqslant|x|^{d_1}]$. Then N_1 n^{d_1}-computes f; furthermore, for some polynomial q, we have

$$\exists^\infty x\in D,\ \exists x{:}\ |z|\leqslant|x|^{d_1}\ [N_1 \text{ on } \langle x, z\rangle \text{ yields } f(x) \text{ within } q(|\langle x, z\rangle|) \text{ steps}].$$

This contradicts the assumption that D is a $(n^{d_1}, poly)$-hard domain of f.

(ii)→(i): Let d_2 be a constant that witnesses (K-i) for $f(D)$, and let d_f be a constant such that $\forall x\in Dom(f)\ [|x|\leqslant|f(x)|^{d_f}]$. Suppose by way of contradiction that D is not

a polynomially hard domain of f; thus, there exist a constant d_2, a polynomial q, and a transducer N_f such that

$$\exists^{\infty} x \in D, \exists z: |z| \leqslant |x|^{d_2/3d_f} [N_f \text{ on } \langle x, z\rangle \text{ yields } f(x) \text{ within } q(|\langle x, z\rangle|) \text{ steps}].$$

Recall that $D \subseteq Dom(f) \subseteq 0^*$. Hence, there exist infinitely many $n_1, n_2, \ldots$ and $z_1, z_2, \ldots$ such that for every $j \geqslant 1$, $|z_j| \leqslant n_j^{d_2/3d_f}$ and N_f on $\langle 0^{n_j}, z_j\rangle$ yields $f(0^{n_j})$ within $q(|\langle 0^{n_j}, z_j\rangle|)$ steps. For every $j \geqslant 1$, let $y_j = f(0^{n_j})$. Then it is easy to define a machine N_2 and a polynomial p_2 such that N_2 on input $\langle n_j, z_j\rangle$ yields y_j within $p_2(|\langle n_j, z_j\rangle|)$ steps for every $j \geqslant 1$. Note that for some constant c and sufficiently large j, $|\langle n_j, z_j\rangle| \leqslant c(\log n_j + |z_j|) \leqslant c(\log n_j + n_j^{d_2/3d_f}) \leqslant n_j^{d_2/2d_f}$ since we are using a binary encoding of integers and assuming that $\forall x, y [|\langle x, y\rangle| \leqslant c_0(|x| + |y|)]$. Thus, for infinitely many j, N_2 outputs $y_j \in f(D)$ from an input of length $\leqslant n_j^{d_2/2d_f} \leqslant |y_j|^{d_2/2}$ within polynomial time. Here, without loss of generality, we can assume that our universal machine N_u can simulate N_2 within polynomial time relative to N_2's running time if the code of N_2, whose length is a certain constant d, is given. Hence, for some polynomial p, $\exists^{\infty} y \in f(D)$ $[y \in \mathrm{K}[n^{d_2/2+d}, p(n)]]$; therefore, $\exists^{\infty} y \in f(D)$ $[y \in \mathrm{K}[n^{d_2}, p(n)]]$. This contradicts the assumption on d_2. □

Corollary 4.7. *The following three statements are equivalent:*

(i) *there exists an infinite set in* PSPACE *satisfying* (K-i);

(ii) *there exists a polynomial-space computable and honest function from* 0^* to Σ^* *that has an infinite polynomially hard domain in* PSPACE; *and*

(iii) *there exists a polynomial-space computable and honest function from* Σ^* *to* Σ^* *that has an infinite polynomially hard domain in* PSPACE.

Proof. (i)→(ii): Let X be an infinite set in PSPACE that satisfies (K-i). Define f from 0^* *to* Σ^* as follows: for every $n \leqslant 0$, $f(0^n) = first(X; n)$. Define D as $Dom(f)$. Then f is a polynomial-space computable and honest function, and D is in PSPACE. Clearly, $f(D)$ is $First(X)$, thus satisfying (K-i). Hence, it follows from Proposition 4.6 that D is a polynomially hard domain of f.

(ii)→(iii) and (iii)→(i): Clear and omitted. □

Property (K-ii) simply states that X has at least one element in $\Sigma^{n \leqslant \cdot \leqslant p(n)}$ for some polynomial p and all n. We name this property "length-dense". That is, a set A is called *length-dense* if $\exists q$: polynomial, $\forall n \geqslant 0$, $\exists x$ $[n \leqslant |x| \leqslant q(n) \wedge x \in A]$. Note that the length-density notion is different from ordinary density; for example, there exists a set A of exponential density, i.e. $\| A^{\leqslant n} \| \geqslant 2^n$, that is not length-dense at all.

It is clear from the definitions and the above observations that we can restate condition (K) as follows:

(K) There exists a PSPACE-computable and honest function f from Σ^* to Σ^* that has a length-dense polynomially hard domain in PSPACE.

4.3. Sufficient conditions for (K)

In order to get better understanding of condition (K), we investigate the possibility that (K) holds. We will show two sufficient conditions for (K).

The first one concerns the problem of whether PSPACE possesses "super-polynomial time" computational power. A measure function t is called *super-polynomial* if $t(n) > n^c + c$ for all constant $c > 0$ and almost all $n > 0$.

Proposition 4.8. *If there exists a super-polynomial running time t such that* $\mathrm{DTIME}(t) \subseteq \mathrm{PSPACE}$, *then* (K) *holds.*

(The idea of the proof is stated in [3]; thus, we omit the proof.)

The next sufficient condition concerns "one-way functions". In this paper, by a *one-way function* we mean an honest and one-to-one (not necessarily total) function f such that f is polynomial-time computable and f^{-1} is not polynomial-time computable. In cryptography, for a feasible and secure cryptosystem, we need a one-way function satisfying several requirements. Here, we formally state some of such requirements and define a necessary condition for "desirable" one-way functions. Then we show that the existence of a one-way function satisfying such a condition implies condition (K).

Let g be any polynomial-time computable function. In order to simplify our discussion, we assume that g is total, honest and one-to-one although this assumption is not essential. We claim that every "desirable" one-way function g should at least satisfy the following condition:

(*5) g^{-1} has a length-dense polynomially hard domain in PSPACE.

A condition similar to the one above has been discussed in several contexts [9]. We explain our motivation of (*5), taking "public-key cryptosystem (PKCS)" [8] for example. What follows is the theoretical framework of PKCS established by Grollmann and Selman [9]. In a PKCS each participant A of the cryptosystem has one pair of keys $(k_1^{(A)}, k_2^{(A)})$, where $k_1^{(A)}$ is made known to the public, i.e. the *public key*, while $k_2^{(A)}$ is kept secret, i.e. the *private key*. This pair of keys has to be prepared prior to communication; such a preparation is carried out using some given polynomial-time algorithm G. More specifically, a participant A, who wants to generate his pair of keys, first randomly generates some seed $x^{(A)}$ and computes $(k_1^{(A)}, k_2^{(A)})$ from $x^{(A)}$ using G, i.e. $\langle k_1^{(A)}, k_2^{(A)} \rangle = G(x^{(A)})$. When a transmitter wants to send a message to A, he encrypts the message using A's public key $k_1^{(A)}$ and send it to A. Then the receiver A uses his private key $k_2^{(A)}$ to decode the message sent by the transmitter.

Let g be a function mapping every x to the first component of $G(x)$. For the sake of simplicity, we assume that g is total, honest and one-to-one. Since G is polynomial time, g is polynomial-time computable. On the other hand, we do not want g^{-1} to be

polynomial-time computable. If otherwise, one can compute $x^{(A)}$ from public key $k_1^{(A)}$, thereby deciphering every secret message sent to A. Thus, we require that g is a one-way function. In addition to that, for a feasible and secure cryptosystem, the following properties are desired for g.

(a) Since the public keys are used for a relatively long time, we want them to be "durable". We measure degree of durability by the intractability of computing g^{-1} when some bits of additional information are given. For example, a public key $k^{(A)}$ is regarded as *insecure* if its seed $x^{(A)}$ $(=g^{-1}(k^{(A)}))$ is easy to compute from $k^{(A)}$ and $|x^{(A)}|^{0.1}$ bits of additional information. Almost all elements of polynomially hard domain of g^{-1} are considered as secure public keys.

(b) Note that every g is easy to invert at infinitely many points. For example, there exists a polynomial-time algorithm that computes g^{-1} for every element of $g(0^*)$. Thus, not all the public keys are secure. Then we need a feasible algorithm that tells us whether a randomly generated seed produces a secure public key or not. We can formalize this notion by a condition that g^{-1} has a polynomially hard domain that is in a low complexity class such as P, NP, or PSPACE.

(c) We need a lot of secure public keys, i.e. we need a polynomially hard domain of high density. For example, it is desirable to have secure public keys at different lengths so that we can easily increase the security by using keys of longer length. We can formalize this requirement by a condition that g^{-1} has a length-dense polynomially hard domain.

Thus, we have proposed condition (*5), which summarizes the above requirements (a), (b), and (c).

Note that for every one-way function, its inverse is polynomial-space computable. Thus, the following proposition is immediate from Corollary 4.7.

Proposition 4.9. *If there exists a one-to-one, honest and polynomial-time computable function g satisfying* (*5), *then* (K) *holds.*

Now, since each of the above two condition implies (K), and (K) implies the separation, we have the following separation results.

Corollary 4.10.

(1) *For any super-polynomial running time t, if* $\mathrm{DTIME}(t) \subseteq \mathrm{PSPACE}$, *then* $\leqslant_T^P$*-completeness differs from* $\leqslant_m^P$*-completeness in* PSPACE.

(2) *If there exists a one-to-one, length-increasing and polynomial-time computable function g such that* g^{-1} *has a length-dense polynomially hard domain in* PSPACE, *then* $\leqslant_T^P$*-completeness differs from* $\leqslant_m^P$*-completeness in* PSPACE.

Remark. A rather simple argument from [22] directly proves (1).

5. Concluding remarks

We have proved the difference between $\leqslant_T^P$-completeness and $\leqslant_m^P$-completeness in PSPACE from several assumptions. Our strategy is to construct a set C and a sparse set S in PSPACE such that $C \oplus T$ is $\leqslant_T^P$-hard for PSPACE and C is not $\leqslant_m^P$-hard for PSPACE; then the key lemma – Lemma 3.4 – proves the existence of a set that is $\leqslant_T^P$-complete and is not $\leqslant_m^P$-complete in PSPACE. Here a question of interest is whether this strategy is useful for proving the difference for other complexity classes such as NP, Σ_k^P, etc. Note that a lemma similar to Lemma 3.4 is provable for every Σ_k^P, $k \geqslant 1$. On the other hand, we have used the high computational power of PSPACE in order to construct sets like C and S.

Acknowledgment

The authors are grateful to Professor R. Book for his support and encouragement. They thank Professor E. Allender and Professor K. Ko for pointing out errors in an earlier version, and also thank referees for many suggestions that helped us very much to improve the presentation of this paper.

References

[1] L. Adleman and K. Manders, Reducibility, randomness, and intractability, in: *Proc. 9th ACM Symp. on Theory of Computing* (ACM, 1977) 151–163.
[2] L. Adleman and K. Manders, *Reductions that lie*, in: *Proc. 20th IEEE Symp. on Foundations of Computer Science* (IEEE, New York, 1979) 397–410.
[3] E. Allender, Some consequences of the existence of pseudorandom generators, *SIAM J. Comput.*, to appear.
[4] E. Allender and R. Rubinstein, P-printable sets, *SIAM J. Comput.* **17** (1988) 1193–1202.
[5] J. Balcázar, J. Díaz and J. Gabarró, *Structural Complexity I* (Springer, Berlin, 1988).
[6] R. Book, D. Du and D. Russo, On polynomial and generalized complexity cores, in: *Proc. 3rd Structure in Complexity Theory Conference* (IEEE, New York, 1988) 236–250.
[7] S. Cook, The complexity of theorem proving procedures, in: *Proc. 3rd ACM Symp. on Theory of Computing* (ACM, 1971) 151–158.
[8] W. Diffie and M. Hellman, New direction in cryptography, *IEEE Trans. Inform. Theory* **IT-22** (1976) 644–654.
[9] J. Grollmann and A. Selman, Complexity measures for public-key cryptosystems, in: *Proc. 25th IEEE Symp. on Foundations of Computer Science* (IEEE, New York, 1984) 495–503; the revised version appeared in *SIAM J. Comput.* **17** (1988) 309–335.
[10] J. Hartmanis, Generalized Kolmogorov complexity and the structure of feasible computations, in: *Proc. 24th IEEE Symp. on Foundations of Computer Science* (IEEE, New York, 1983) 439–445.
[11] R. Karp, Reducibility among combinatorial problems, in: R. Miller and J. Thatcher, eds., *Complexity of Computer Computations* (Plenum Press, New York, 1972) 85–104.
[12] K. Ko, On self-reducibility and weak *P*-selectivity, *J. Comput. System Sci.* **26** (1983) 209–221.
[13] K. Ko and D. Moore, Completeness, approximation and density, *SIAM J. Comput.* **10** (1981) 787–796.

[14] L. Longpré and P. Young, Cook is faster than Karp, in: *Proc. 3rd Structure in Complexity Theory Conference* (IEEE, New York, 1988) 293–302.
[15] N. Lynch, On reducibility to complex or sparse sets, *J. Assoc. Comput. Mach.* **22** (1975) 341–345.
[16] S. Mahaney, Sparse complete sets for NP: solution of a conjecture of Berman and Hartmanis, *J. Comput. System Sci.* **25** (1982) 130–143.
[17] A. Meyer and M. Paterson, With what frequency are apparently intractable problems difficult?, Tech. Report MIT/LCS/TM-126, Massachusetts Institute of Technology, Cambridge, 1979.
[18] E. Ukkonen, Two results on polynomial time Turing reductions to sparse sets, *SIAM J. Comput.* **12** (1983) 580–587.
[19] L. Valiant and V. Vazirani, NP is as easy as detecting unique solutions, *Theoret. Comput. Sci.* **47** (1986) 85–93.
[20] U. Vazirani and V. Vazirani, A natural encoding scheme proved probabilistic polynomial complete, *Theoret. Comput. Sci.* **24** (1983) 291–300.
[21] O. Watanabe, On the Structure of Intractable Complexity Classes, Dr. Eng. Dissertation, Tokyo Institute of Technology, Tokyo, 1987.
[22] O. Watanabe, A. comparison of polynomial time completeness notions, *Theoret. Comput. Sci.* **54** (1987) 249–265.
[23] O. Watanabe, On $\leqslant^{P}_{1\text{-tt}}$-sparseness and nondeterministic complexity classes, in: *Proc. 15th Internat. Collo. on Automata, Languages and Programming*, Lecture Notes in Computer Science, Vol. 317 (1988) 697–709.